コマンドラインを使い、PDFをテキストにする

朝日新聞国際報道部・西尾 能人
yoshito.nishio+JNPC@gmail.com

0. はじめに

ふだんはマウスを使って直感的に操作しているパソコン。でも、SF映画で、「黒い画面」に命令を打ち込んで操作するシーンを見たことがあるはず。近未来的とか、プロっぽい、とかいったイメージが託されている……のかも。
昔ならともかく、実は今でもこの「黒い画面」を使う人がいる。どんなメリットがあるのか、少しだけ体験する。その例として、PDFからテキストを抽出してみる。
下記の事前準備が難しかった方は、きょうは見るだけにして、「こういうものがある」ということだけ記憶に留めておく。大量のファイルに同じ処理をしたい、というニーズに直面したときに思い出そう。技術に詳しい人に声を掛けて、手伝ってもらえばよい。

※「黒い画面」と言っても、設定次第では、白い背景だったり青い背景だったりする。

1. このコマの目標

- LinuxやMacのコマンドラインを知る
- コマンド操作で、大量のPDFをテキストに変換する

2. このコマでやること

- 1つのことをやる「コマンド」を組み合わせることで、複雑な処理をする
- 大量のファイルに同じ処理を適用する
- 文字情報を持っているPDFからのテキスト抽出。Popplerというツール群に含まれている「pdftotext」を使う

3. 可能なら事前準備を

ただし、**決して無理しない**こと!
自信がない場合は、詳しい方に手伝ってもらうか、インストールしないままで見学を。

▶Windowsをお使いの方のみ

3.1. Windows PCへのWSL2とUbuntuのインストール

UbuntuというのはLinuxの一種。以下の手順で、デフォルトでインストールされる。

1. Windowsキー + Xキーを押し、メニューから「Windows PowerShell(管理者)」を選ぶ。または、管理者権限ありで、PowerShellかcmd.exeを実行する。
2. 開いた窓に、以下のコマンドを打ち込み、改行で実行する。

```
wsl --install
```

あとは、画面の指示に従う。
参考：Microsoftのサイトの手順案内

3. ペンギンマークのアイコンで、Linuxの端末が開く。初回のみ、ユーザー設定が必要。
参考：Microsoftのサイトの手順案内
4. もし日本語環境になっていないようなら、そのときのみ、以下を実行
※今でもこれが必要か不明。どなたか教えて下さい。

```
sudo apt install language-pack-ja
sudo update-locale LANG=ja_JP.UTF-8
```

▶Windowsをお使いの方のみ

3.2. Windows PCへの「pdftotext」のインストール

1. Linuxのターミナルで、以下のコマンドを実行。

```
sudo apt install poppler-utils
```

▶Macをお使いの方のみ

3.3. Macへの「pdftotext」のインストール

1. Macのターミナルで、以下のコマンドを実行（Homebrewを使っている場合）。「tree」コマンドが不要なら、１行目だけでよい。

```
brew install poppler
brew install tree
```

4. このコマでは扱わないこと

- 文字情報を持たない「画像」状態のPDFの処理
- 文字情報を持っていて、かつ表形式の場合。おすすめは「tabula」。ブラウザを使うが、スタンドアローンで動作する

この２点は、スポーツ報知・田中さん、ＮＨＫ・山本さんからご紹介があります。

- Windows PCにLinux環境を構築する方法
- pdftotext（Poppler）のインストールがうまく行かない場合の対処

いずれも、社によってセキュリティ周りの設定やOSのバージョンが異なり、いっせいにご案内するのは難しいため。
「コマンドラインを使えばできるかも」ということが頭の片隅に残れば、必要になったときに、社内の詳しい方に相談できる。とりあえずは、それで十分。

- GUI画面で、ファイル拡張子を表示させる方法（ご自身で調べてみて）

5. フォルダの準備

練習用のファイル一式を、GitHubに置いてある。https://github.com/nishioWU/JNPC2205からダウンロードを。
右上の緑の「code」から「Download ZIP」を選ぶ。ダウンロード後は必ず解凍し、デスクトップなど適当な場所に「JNPC2205-main」というファイルごと移しておく。
このコマでは、このフォルダの中だけで作業する。

6. コマンドラインを試す

6.1. Linux 端末の起動

Windows PCでは、デスクトップに準備した「JNPC2205-main」のフォルダを、Shiftキーを押しながら右クリックする。すると、普段の右クリックでは出てこない「Linux端末をここに開く」という項目が見えるので、それを選ぶ（ここで開く、に直っているかも）。「bash」というものが立ち上がる。
Mac の場合は、フォルダを２本指タップし、「サービス」から「フォルダに新規ターミナル」を選ぶ。
または、「ターミナル」を起動してから、その画面に「cd （半角スペース）」と打ち、きょうの作業場所となる「JNPC2205-main」のフォルダを黒い画面にドラッグしてから、return（またはenter）キーを打つ。最近のMacなら 「zsh」というものが立ち上がる。古いOSからアップデートしている場合は、bashのこともある。どちらであるかは、ターミナル上部に表示されているはず。
bash と zsh では挙動が違う部分があるが、本日扱う範囲では、それほど大きな差はない。

※しかし私は挙動の違いにはまってしまい、中日新聞・松波さんに助けてもらった。findの動作テストにも協力いただき深謝。

6.2. 使い方の基本

黒い画面に文字を打ち込み、最後にEnterキーを押すと、コマンドが実行される。きちんと実行された場合には、特段のメッセージは表示されない。また、危険な操作の場合も、確認・念押しはされずに、実行されてしまうので、要注意。Undoで元に戻すこともできない。

コマンドの履歴は↑キーでたどれるので（行き過ぎて戻るときは↓）、少し打ち替えて実行する場合に便利。打ち替える必要がなく、同じコマンドをもう一度実行する場合は!!。履歴の中から検索するならCtrl + R。文字を打ち込むにつれて絞り込まれる「インクリメンタルサーチ」が使える。履歴を打ち替えて再利用するならEscだったり←や→だったり（挙動に差あり）、使わずに抜けるならCrtl + Gを。
ファイル名やコマンドは、ほかと区別のつく部分まで入力してTabキーを押せば、自動補完される。区別がつかない場合には、音が鳴るので、もう少し先まで打ち足せばよい。またはTabを２度押すと、候補の一覧が表示される。zshなら、Tabを押すたびに、次の候補が入力される。
入力を中断したいときはCtrl + C。実行中のコマンドをとめたいときも、同じくCtrl + C。画面をいったんクリアしたいときは、Ctrl + L。フリーズしたら、Ctrl + Dか、マウス操作で黒い窓を閉じて、やり直し。
このほか、Ctrl + Aで行頭に移動、Ctrl + Eで行末に移動、Ctrl + Kで行末まで削除・切り取り、Ctrl + Uで行頭まで削除・切り取り（zshでは１行丸ごと削除）、Ctrl + Yで貼り付け、のショートカットが使える。

なお、Linuxの世界では、Windowsと違って大文字と小文字は区別され、別物として扱われることに留意を。ファイルが見つからない旨のエラーが出るときも、これが原因のことがある。文字コードも、

Windows標準のSHIFT-JISではなくUTF-8。なので、テキストファイルであるにも関わらず、開いたとき文字化けしていたら、コードを選び直して再度開くか、コードを変換してやることになる。

6.3. コマンドとオプション

まず、今日使うフォルダの構成を見てみる。「tree」コマンドを使う。黒い画面に

```
tree
```

と入力して、Enterを押すと実行される。見慣れたGUIではないが、ファイルとフォルダが区別して表示されているのが分かるだろうか。
あるフォルダ内のファイル一覧を見るには、「ls」コマンドを使う。やはり、黒い画面に

```
ls
```

と入力し、Enterを押すと実行される。
今いる場所から別のフォルダに移動するときは、

```
cd フォルダ名
```

逆に元のフォルダに戻るときには、

```
cd -
```

と入力する。1階層上に行くときには、

```
cd ..
```

とする。よく使う。「.」1個は今いるフォルダという意味で、2個だと1階層上を指す。
動作を少し変えたいときには、ハイフンでオプションを足す。
例えば、ls コマンドでファイル名が縦に1つずつ表示されるようにしたければ、半角数字の「1」を添えて

```
ls -1
```

とする。
同じ意味のオプションでも、長短２通りの指示の出し方があることが多い。短いオプションのときはハイフン１つ、長いオプションのときはハイフン２つが基本だが、長いオプションでもハイフン１つのときも稀にある。
詳しい使い方を見たいときは、以下のように打つ。

```
ls --help
```

ファイルを１つずつ縦に並べるなら「-1」（数字のイチ）、詳しい情報を見たいときには「-l」（longのエル）、コンマで区切ってほしければ「-m」、ファイルかフォルダか区別したければ「-F」（色分けされていないときに助かる）、拡張子順なら「-X」（大文字のエックス）、タイムスタンプ順なら「-t」、順番をひっくり返したいときは「-r または --reverse」など、いろいろあることが分かる。

```
ls -1
ls -r
ls -F
```

などと入れてみよう。
オプションは組み合わせることもできて、

```
ls -1tr
```

とすれば、「１つずつ」「タイムスタンプ」「逆順」なので、最新のファイルから順に表示することができる。

なお、

```
man コマンド名
```

とすると、ヘルプよりも詳しいマニュアルが読める。Macのzshでは、デフォルトではヘルプがオンになっていないので、こちらの方法で。

6.4. よく使うコマンド

コマンドは無数にあるが、まず使うのは

- exit（黒い画面を閉じて、GUIの世界に戻る）
- pwd（今どのフォルダにいるかを表示する）
- ls（今いるフォルダのアイテムを表示する）
- less（テキストファイルを 1 ページずつ表示する。ページ送りはスペースキー、終了は Q 。ファイル内の検索もできる。複数のファイルを同時に開いた場合は、次のファイルへの移動は : n 、前のファイルに戻るときは : p ）
- cat（複数のファイルをつなげる。または、テキストファイルを表示する）
- man（コマンドのマニュアルを表示する）
- cp（ファイルやフォルダをコピーする）
- mv（コピーせずに移動する。または名前を変更する）
- grep（テキストファイルを検索する。「正規表現」というものが使える）
- sort（テキストをソートする）
- uniq（ソート済みのものについて、ダブりを削除する）
- wc（スペースで区切られた単語数や、行数を数える）
- cd（別のフォルダに移動する）
- mkdir（フォルダを新しく作る。エクスプローラーやファインダーで作っても同じこと）
- find（条件にあうファイルを探す）
- xargs（検索結果などを「引数」として次のコマンドに渡して実行する）

あたり。その都度、使い方を調べて入力すればよい。私もよく使うオプション以外は頭に入っていない。

6.5. パイプライン、リダイレクトとワイルドカード

個々のコマンドは、単一のことをしっかりやる、という設計思想で作られている。回転寿司の一皿みたいなもので、いざ、まとまった作業をしようとすると、物足りない。そこで、必要に応じて複数のコマンドをつなげて使う。Aで処理した結果をBに渡して処理させ、Cというファイルに書き込む、というように。

あるコマンドを実行した結果を、別のコマンドに渡すときには、 | （パイプ、パイプライン）を使う。 Shift キーを押しながら ¥ マークのキーも押すと、出る。
別のコマンドに渡すのではなく、ファイルに書き込むときには、 > か > > （リダイレクト）を使う。 1 個なら上書き、２個だと追記になる。存在しないファイル名を指定すると、その名前で新たにファイルが作られる。
それから、ファイル名の指定にはワイルドカードが使える。 * は任意の文字列（文字なしも含む）、 ? は任意の 1 文字。このほかにもあるが、複雑なので、とりあえず２つだけ覚えておけばよい。

参考：Linuxのワイルドカードと正規表現についてのサイト（ためになるが難しい）

コマンドを複数つなげて使い、ワイルドカードを活用することで、大量のファイルを一括処理することが

できる。それが、あえてマウスを使わずにコマンドを使う理由。たくさんの文書やデータと格闘するときに、役立つことが多い。

6.6. grepで絞り混み、wcでカウント

では、練習のため、cdコマンドで「TEXT」フォルダに移動を。 例えば、

```
grep -n 東区 text*.txt > result.txt
```

こうすると、①今のフォルダの中にある「text」で始まる名前のテキストファイルすべてから②「東区」という文字列を含む行を探して③「result.txt」というファイルに書き込むーーという命令になる。
result.txtというファイルがなければ作るが、もしあったら、上書きされてしまう。上書きでなく追記したいなら、「>」を「>>」に変えてやる。
grepの「-n」は、行番号を振るオプション。
今は検索対象のファイルはたった４つしかないが、ワイルドカードで指定しているので、いくつに増えても同じ命令で事足りる。
ところで、「text」と全部打たずに、「t」だけ打って Tab を押せば、自動補完されることをお忘れなく。そのとき音が鳴るのは、「text」で始まるファイルが複数あるのでこの先は特定できませんよ、と知らせてくれているわけ。

次に、「南区」を検索して、先ほどのファイルに追記してみよう。 ↑ で今のコマンドを呼び出して編集してやる。「東区」を「南区」に、「>」は「>>」に直す。

```
grep -n 南区 text*.txt >> result.txt
```

東と南がどちらが多いか数えることもできる。やはり、使ったコマンドを呼び出して編集して

```
grep 東区 text*.txt | wc -l
```

と、

```
grep 南区 text.txt | wc -l
```

を実行して比べてみよう（わざわざ「-n」を削らなくてもよい）。最後につけたオプションは、lineのエル。語数ではなく行数をカウントする、という指示。

コマンドが正常に実行されていれば特段のメッセージが出ないのは、このようにパイプでつなげたり、ファイルに書き込んだりする際に、そのほうが便利だからだ。なお、コマンドの実行結果とエラーメッセージは別物として管理されているため、エラーメッセージだけをログファイルに追記する、ということもできる。

6.7. ファイルを移動する

では、次に進むために、もう不要になったファイルを掃除する。gomibakoというフォルダを作って

```
mkdir gomibako
```

そこに、tで始まるファイル全てを移動させてしまおう。

```
mv t* gomibako
```

result.txtももう不要なので、

```
mv result.txt gomibako
```

で移動する。ファイル名は、全部打たずに自動補完するのが楽。ファイル名で選り分ける必要がなく、テキストファイルをみな移動させると考えて、「mv *.txt gomibako」でもよい。慎重かつ臨機応変に。

7. pdftotext を試す

7.1. 自分の居場所を移動する

今度は、「PDF」フォルダの中の「source」フォルダに移動を。いったん上の階層に行ってから、また降りていくようにする（横に移動することはできない）。1 階層上に行くには、

```
cd ..
```

先ほどまでいたフォルダに戻る、という指示の仕方なら

```
cd -
```

で親フォルダに戻り、そこで

```
cd PDF
cd source
```

で降りていく。打ち間違いを防ぐためにも、適宜Tabキーを押して自動補完しよう。一度に

```
cd PDF/source
```

でもよい。

7.2. pdftotextの使い方

いよいよ、pdftotextを使う。これは、Popplerというツール群の中のコマンドで、文字情報を持っているPDFから、テキストだけを抜き出してくれる。

```
pdftotext ファイル名
```

とすると、「abc.pdf」というファイル名のPDFを、拡張子だけ変えた「abc.txt」というテキストファイルに変換する。

参考：Poppler 公式サイト
pdftotextの使い方は、以下が詳しい。
参考：西村めぐみさんの記事
西村さんの著書「Linux+コマンド入門 ——シェルとコマンドライン、基本の力」は、Linuxのコマンド全般について役に立つ。pdftotextには触れていない。なお、オプションは主なものしか載っていないので、一度は「コマンド名 --help」で、使いたいコマンドのヘルプを見るべきだ。

pdftotextコマンドの出力結果をテキストファイルに書き込むのではなく、画面に出すだけでよければ、

```
pdftotext ファイル名 -
```

のように、最後にハイフンを 1 つ付ける。ページ区切りが煩わしければ、「-nopgbrk」オプションを付

ける。画面をさーっと流れて行ってしまう長い文書なら、

```
pdftotext ファイル名 - | less
```

のように、パイプでlessに渡してやる。
「PDF_番号.pdf」というファイルがいくつかあるので、ちゃんとテキストに変換されるか、見てみよう。くどいけれど、ファイル名入力はなるべく自動補完で。

7.3. pdftotextの「文法」の注意点

ところで、１つずつではなく、全部まとめて調べてみよう、と考えて

```
pdftotext PDF_*.pdf -
```

とか、さらに汎用的にPDFを全部捕まえると考えて

```
pdftotext *.pdf -
```

を試みた方はいないだろうか。私も同じことを考えた。
しかし、コマンドのヘルプを見ると、pdftotextの文法は「pdftotext [options] <PDF-file> [<text-file>]」となっている（[ ]内は省略可能）。ファイル名は１つまたは２つしかコマンドの後には置けず、１つ目が変換すべきPDF、２つ目がある場合はそれは抽出したテキストの書き込み先、という仕様だ。なので、「PDF_*」の条件に当てはまるファイルがたまたま１つなら正常動作するが、偶然２つあった場合は、２つ目は変換元ではなく書き込み先になってしまう（すでにファイルが存在している場合は上書きされる。catコマンドで中を見るか、拡張子を.txtに変えてWindowsのメモ帳で読むかして、確かめてみて）。ファイルが３つ以上見つかった場合はエラーして動かないので、ヘルプ画面が表示される、ということになる。

では、ファイルを１つずつ処理するしかないのか？ そもそも、１つずつテキストに変換するのでよいなら、Adobe Acrobat Readerでできる。文書を開いた後、「ファイル」→「その他の形式で保存」→「テキスト」を選べば済む。
なのにわざわざコマンドラインを使う理由は、大量のファイルを一気に処理したいからだった。どうすればよいだろうか。

7.4. おまけ：Ubuntuだけの楽なやり方

Wondows PCのWSL2でデフォルトでインストールされるUbuntuの場合、pdftotextをインストールすると、lessコマンドでいきなりPDF内のテキストが読めるようになる。気を利かせて、そういう仕様にしているようだ（時事通信・川上さん、ご教示多謝です）。lessは一度に複数のファイルを指定しても問題なく動くので、こうすればよい。

```
less 名前の目印*.pdf > 出力ファイル名.txt
```

この 1 行だけで、今いるフォルダの中にある、ある文字列で始まる名前のpdfファイルすべてを、テキストに変えて一本につなげ、出力先のテキストファイルに保存する、という指示になる。具体的には、

```
less *.pdf > joined_text.txt
```

などとコマンドを打てばよい。Ubuntuを使っている方には朗報。

8. 大量ファイルの一括処理

8.1. xargsでコマンドを組み合わせる

そこで、xargsというコマンドを使うことにする。これは、直前のコマンドの結果を、次のコマンドの好きな位置に埋め込むのに使える。pdftotextに、一度に 1 つずつファイル名を渡すようコントロールするのに、うってつけだ。「-I」（大文字のアイ）というオプションに、`@`でも`{}`でも何でもよいので目印の記号を付けて、こんなふうに使う。

```
前のコマンド | xargs -I{} 次のコマンド 入れたい場所に{}
```

この例では、{}を書いた場所に、「前のコマンド」の実行結果が 1 つずつ入る。それが尽きるまで、「次のコマンド」を繰り返して実行する。だから、pdftotextに 1 つずつファイルを渡して、抽出したテキストを画面に表示するには、

```
ファイルを探すコマンド | xargs -I{} pdftotext {} -
```

とすればよい。

8.2. findでファイルを探す

PDF形式のファイルを探すには、findコマンドを使うことにする。lsとは違い、今いるフォルダだけでなく、それより下の階層をすべて探してくれる。

```
find 探す場所 -name '*.pdf'
```

のように使う。
探す場所は、「.」とすれば、今いるフォルダを起点に、ここより下の階層も全部、という指示になる。
-name（厳密にはオプションではなく、評価式というものの一部で、ハイフンは1つ）。もし、ファイル名の大文字・小文字を区別したくなかったら、-inameオプションにする。その後は、「こんな名前のファイル」という検索条件を、シングルクオート（アポストロフィ。 Shift + 7 で出る）で囲む。
もし、並び順を辞書順ではなく、自然な数字順にしたければ、sortに渡してやる。lsの場合は小文字で「-v」とする（Macのzshのlsだと、-vオプションは別の働きが割り当てられている）のだが、sortは大文字で「-V」とするか、よく似た働きの「-n」とするので、こうなる。

```
find . -name '*.pdf' | sort -V
```

場所の「.」を省略しても、環境によっては動くようだ。
zshでは、lsで「/**/」という書き方を使って深い階層まで探すことができるので

```
ls ./**/*.pdf | sort -V
```

とすることもできる。

8.3. 1行プログラム完成

これを、先ほど試したものにつなげて、

```
find . -name '*.pdf' | sort -V | xargs -I{} pdftotext {} -
```

こうなる。あと一息。画面に流すのではなく、全部をまとめて1つのテキストファイルに保存しておくなら、ファイルへのリダイレクトを足す。「joined_text.txt」というファイル名にするなら、以下のように

なる。ついでに、長い文書の場合にPDFのページ区切りを取り除く、-nopgbrkというオプションも入れた。

```
find . -name '*.pdf' | sort -V | xargs -I{} pdftotext -nopgbrk {} - > joined_text.txt
```

段階的に試しながら、コマンドを履歴から編集して打ち足してきた。いきなりこれを見せられたら、呪文のようで逃げ出したくなるが、今なら何とか分かりそうな気がする。とりあえず、逃げ出しはしない……のではないだろうか。たった 1 行だけではあるけれど、簡単なプログラムがこれで完成だ！

zshの場合は、lsを使って以下のようにしてもよい。

```
ls ./**/*.pdf | sort -V | xargs -I{} pdftotext -nopgbrk {} - > joined_text.txt
```

出力場所を変更したいとき、例えば「今いるフォルダと同じ階層にある、outputというフォルダ」の下にしたいときには、

```
find . -name '*.pdf' | sort -V | xargs -I{} pdftotext -nopgbrk {} - > ../output/joined_text.txt
```

とする。「..」で 1 階層上に上がり、その下にある「output」フォルダに降りて来る、という指定の仕方をしている。

zshの場合は

```
ls ./**/*.pdf | sort -V | xargs -I{} pdftotext -nopgbrk {} - > ../output/joined_text.txt
```

でもよい。

PDFなら何でも、ではなく、テキスト抽出の対象とするPDFを絞り混みたいときは、findの後の' 'の中を変えればよい。'*_2022-??-??.pdf'（ファイル名の末尾に「_2022-XX-XX」というタイムスタンプが入っている）とか、'都道府県別データ*.pdf'（ファイル名が「都道府県別データ」で始まる）といった塩梅。

8.4. PDFごとに変換したいとき

さて、全体を 1 つのテキストファイルにまとめるのではなく、PDFごとに同名（拡張子は「.txt」に変わる）のテキストファイルに出力するのだったら、どうすればよいか。ファイルを検索するまでは一緒だが、ソートもリダイレクトも不要になる。

もとのPDFがあるのと同じフォルダに、それぞれテキストファイルを書き出すとすると、こうなる。

```
find . -name '*.pdf' | xargs -I{} pdftotext -nopgbrk {}
```

zshの場合は、

```
ls ./**/*.pdf | xargs -I{} pdftotext -nopgbrk {}
```

でもよい。

8.5. 階層構造を保ったまま、変換済みテキストをあるフォルダにまとめたいとき

PDFごとに同名のテキストファイルに変換した上で、階層構造を保ったまま、extracted_textというフォルダにまとめたい、ということもあるだろう。 1 行で書くのはたぶん無理なので、最小限のプログラムをサンプルとして示す。WSLでもMacでも動く。

▶ファイル名「p2t.sh」

```
#!/bin/sh
find . -name '*.pdf' | xargs -I{} dirname {} | sed 's/^.//' | xargs -I{}
mkdir -p extracted_text{}
find . -name '*.pdf' | xargs -I{} pdftotext -nopgbrk {}
find . -name '*.pdf' | sed 's/^.//' | sed 's/.pdf$/.txt/' | xargs -I{}
mv .{} extracted_text{}
```

このプログラムのファイルを、使いたいフォルダにコピーする。次に、そのフォルダに移動してから

```
chmod +x p2t.sh
```

と打って、ファイルを「実行可能」にしてやる。「x」は小文字のエックス。そして、

```
./p2t.sh
```

と打つと、自前のプログラムが実行される。先頭についている「./」は、今いるフォルダにあるものだよ、という意味で、これがないと、正規のコマンドとPCが勘違いして別の場所を探しに行ってしまう。これで、今いる場所より下の階層にあるPDFがすべて、テキスト形式に変換されて、extracted_textフォルダの下に抽出される。もともとextracted_textという名前のフォルダがあったら、置き場所も名前も同じテキストは上書きされ、そうでないものは追加されて混ざってしまうので、フォルダ名を変えるか、プログラムを手直しすること。

各行はそれぞれ、
①このファイルは、端末に実行させるためのプログラムだ
②拡張子が.pdfのファイルを探し、その置き場所までのツリー構造をextracted_text傘下に再現せよ
③PDFと同じ置き場所で、テキストファイルに変換せよ
④PDFと同名で拡張子が.txtになっているファイルのみ、extracted_text傘下の対応する位置に移動せよ
――という指示になっている。この手順のうち、③の部分は、上でやったばかりの1行プログラムとまったく一緒であることに、気がついただろうか。

※プログラムを書いたり手直ししたりする場合には、「Word」など文字修飾機能があるワープロソフトではなく、「Visual Studio Code」などのテキストエディタを使う。文字コードはUTF-8で改行はLFのみ（CRなし）に。

8.6. おまけ：画像を抜き出す

余談ながら、Popplerをインストールすると、pdftotextだけでなく、PDFから画像を抜き出すpdfimagesなどほかのコマンドも使えるようになる。「日本記者クラブ.pdf」というPDF中の画像を抜き出し、png形式でないものは変換した上で、「JNPC-xxx.png」というファイル名で保存したい場合は、

```
pdfimages -png 日本記者クラブ.pdf JNPC
```

とする。

9. おわりに

9.1. 枯れた技術ゆえの強み

大量のファイルを処理する場合、最初に対象のファイルを全部まとめてしまってから一度だけ処理する手もあるだろうし、ファイルごとに処理してから最後にまとめるという手順も考えられる。取り組んでいる課題に合わせて、効率がよく、手戻りしない組み立てを考えればよい。

Linuxコマンドは、先人の検証が十分になされている「枯れた技術」なので、技術に詳しい人に訊くか、ネット検索すれば、いろいろと方法はヒットするはずだ。

9.2. 質問は財産、Slackで共有を

もちろん、Slackのチャンネルに質問を投稿いただければ、私のレベルでは分からないことでも、一緒に学び、仕事に使っている仲間が助けてくれることと思う（なので、メールよりはSlackがおすすめ）。
「ここが分からない」と思ったところ、「こうしたいのに」と感じたところは、多くの方に共通するはずだし、質問と解決法はみんなの財産になる。遠慮は無用！

駆け足でお疲れさまでした。時間のあるときにゆっくり、試してみて下さい。